Грокаем алгоритмы. Глава 2. Сортировка выбором

В этой главе

- Вы познакомитесь с массивами и связанными списками - двумя основными структурами данных.
- Объясняются плюсы и минусы обеих структур данных, чтобы вы сами могли решить, что вам использовать.
- Вы изучите свой первый алгоритм сортировки, который поможет в будущем при изучении алгоритма быстрой сортировки

Как работает память

Представьте, что вы пришли в театр и хотите оставить свои личные вещи. Для хранения вещей есть специальные ящики, в каждый из которых помещается один предмет. Если вы хотите положить две вещи, вам потребуется выделить два ящика для этого.

Вы оставляете свои вещи в двух ящиках. Готово! Можно идти на спектакль!

В сущности, именно так и работает память компьютера. Она представляет собой нечто вроде большого шкафа со множеством ящиков, у каждого из которых есть свой адрес.

Каждый раз, когда вы хотите сохранить в памяти отдельное значение, вы запрашиваете у компьютера место в памяти, а он выдаёт адрес для сохранения значения. Если вам понадобиться сохранить несколько значений, то можно воспользоваться массивами или связанными списками. В следующем разделе мы обсудим их преимущества и недостатки.

⚠ **Заметьте**

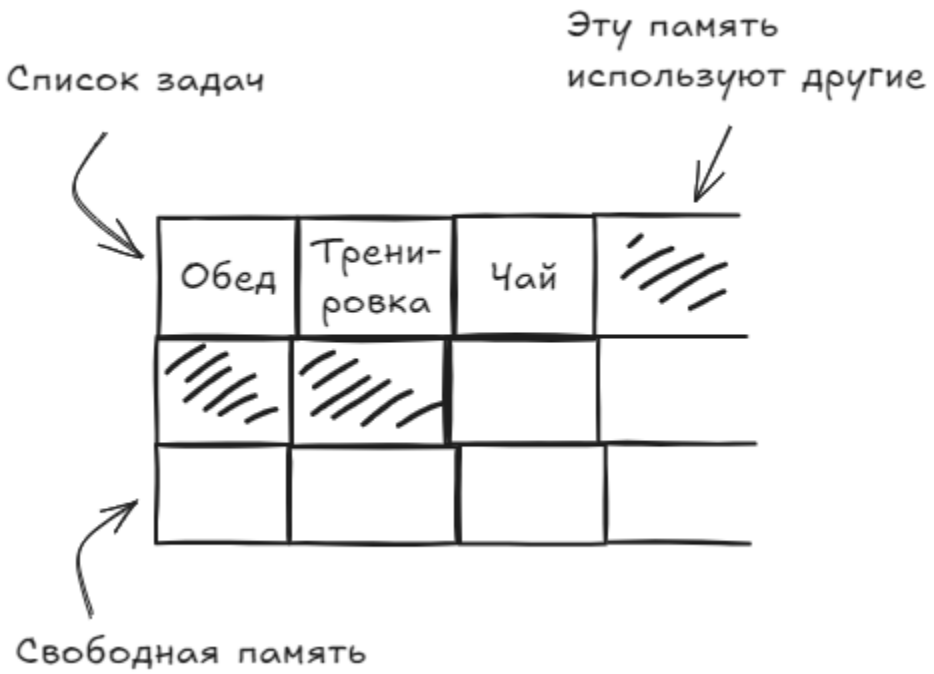
Не существует единственно верного способа сохранения данных на все случаи жизни, поэтому вы должны знать отличие разных способов

Массивы и связанные списки

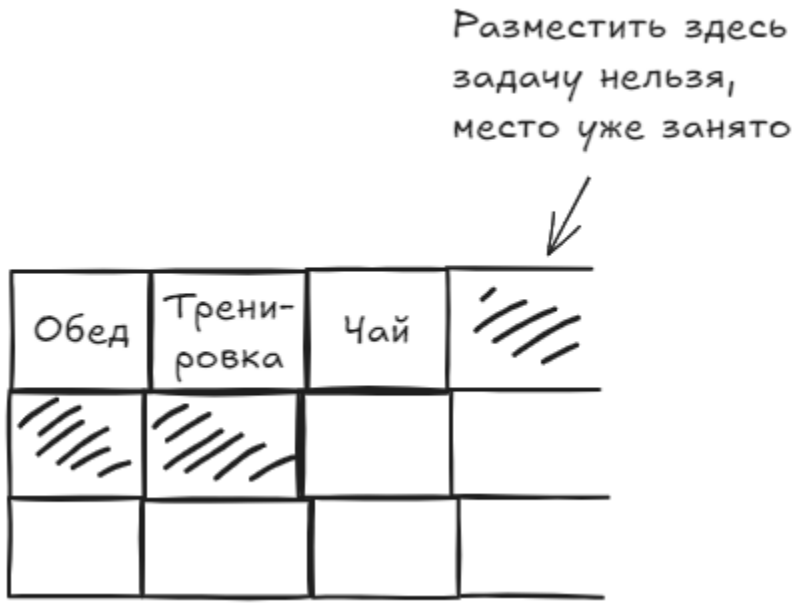
Иногда в памяти требуется сохранить список элементов. Предположим, вы пишите приложение для управления текущими делами. Описание задач в памяти должно храниться в виде списка.

Что использовать - массив или связанный список?

Для начала попробуем разместить сохранить задачи в массив, потому что этот способ более понятен. При использовании массива все задачи хранятся в памяти непрерывно (то есть рядом друг с другом)



Теперь, предположим, что вы захотели добавить четвёртую задачу. Но следующий ящик уже занят - там лежат чужие вещи!



Представьте, что вы пошли в кино с друзьями и нашли места для своей компании, но тут приходит ещё один друг, и ему сесть уже некуда. Приходиться искать новое место, где смогут сесть все. В нашем случае вам придётся запросить у компьютера другой блок памяти, в который поместятся все задачи, а потом переместить их туда

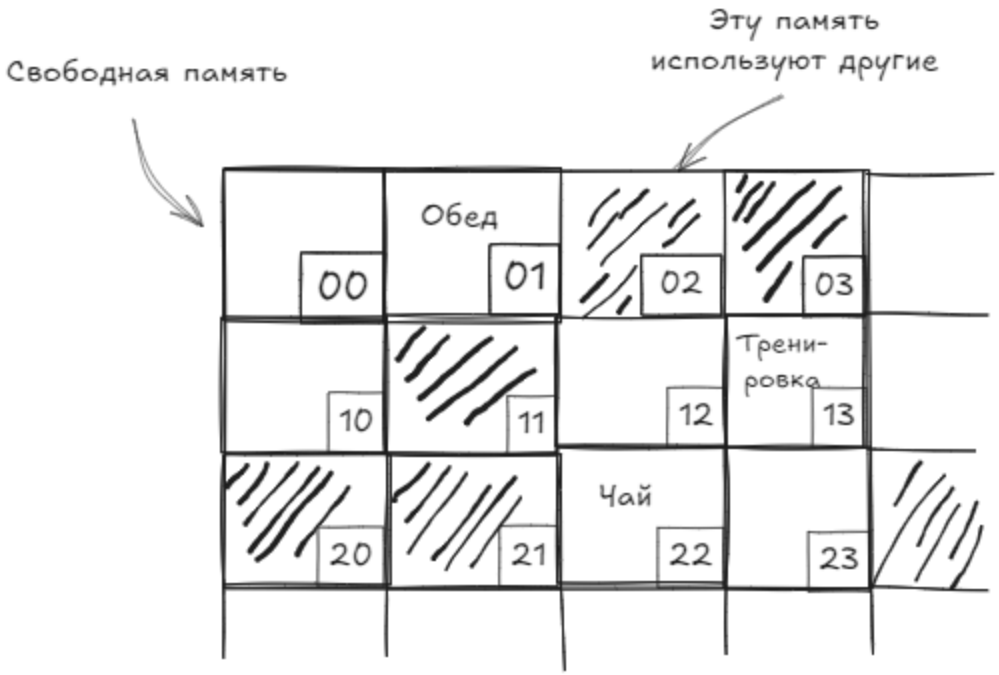
Если вдруг придёт ещё один друг, то места опять не хватит и вам всем придётся перемещаться снова! Сплошная суета. Простейшее решение - "бронирование мест": даже если список состоит из всего 3-х задач, вы запрашиваете место на 10 позиций... просто на всякий случай. Тогда в список можно будет добавить до 10 задач. Это неплохое решение, но есть НО:

- Лишнее место может не понадобиться, и тогда память будет расходоваться неэффективно. Вы её не используете и другим не даёте
- Если в список будет добавлено более 10 задач, перемещать всё равно придётся

В общем, приём неплохой, но не идеальный. Связанные списки решают проблему добавления новых элементов.

Связанные списки

При использовании связанного списка элементы могут находиться где угодно в памяти



В каждом элементе храниться адрес следующего элемента списка. Таким образом набор произвольных адресов в памяти объединяется в цепочку

Всё как в игре "Найди клад". Вы проходите по первому адресу, там написано: "Следующий элемент находится по адресу 123". Вы идёте туда, а там снова находится записка с указанием на следующий адрес итд. Добавить элемент в такой список очень просто: просто разместите его в любой ячейке памяти и расскажите об этом предыдущему элементу.

Со связанными списками ничего перемещать не нужно! Если вернуться к примеру с кино, то вы говорите друзьям: "Ладно, тогда садитесь на свободные места и смотрите фильм". Так и в памяти, если есть свободное место, то вы сможете сохранить данные в связанном списке

Вопрос на подумать

Если связанный список так хорошо справляется со вставкой, то чем тогда хорош массив?

Массивы

На сайтах со всевозможными хит-парадами и "первыми десятками" применяется жульническая схема для увеличения просмотров. Вместо того чтобы вывести весь список на одной странице, они размещают по одному элементу на каждой странице и заставляют вас нажимать на кнопку "Next" для перехода к следующему элементу.

В результате сайту удаётся показать вам рекламу на целых 10 страницах, но нажимать "Next" 9 раз для перехода к первому месту скучно. Было бы лучше, если всё помещалось на одной странице, а вы могли бы просто щёлкнуть на интересное место, для получения дополнительной информации.

Похожая проблема существует и связанных списков. Допустим, вы хотите получить доступ к последнему элементу. Просто прочитать нужное значение не удастся, вместо этого нужно будет по всем элементам.

В итоге имеем:

	Массивы	**Списки**
Чтение	$O(1)$	$O(n)$
Вставка	$O(n)$	$O(1)$
Удаление	$O(n)$	$O(1)$

Вопрос на подумать

Почему у массива и связанного списка именно такие скорости на вставке и удалении?

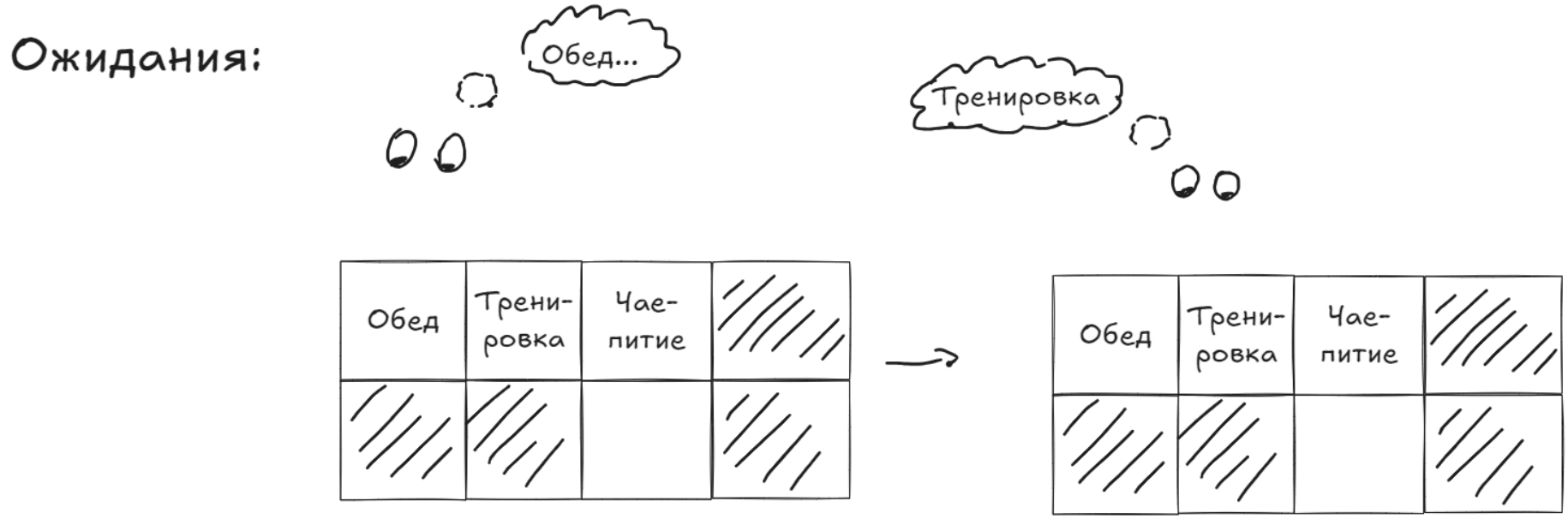
Какая структура данных используется чаще?

Массивы используются часто, поскольку имеет множество преимуществ перед связанным списком. Во-первых, их проще читать. Во-вторых, они поддерживают произвольный доступ.

Всего существует два вида доступа: последовательный и произвольный. При последовательном все элементы читаются по одному, начиная с первого. Связанные списки поддерживают только такой доступ. Многие реальные ситуации требуют произвольного доступа, поэтому массивы часто применяются на практики.

Однако даже безотносительно произвольного доступа массивы работают быстрее, поскольку используют кэширования. На рисунке представлено ваше, возможное, представление о процессе чтения массива и реальный

процесс чтения.



Таким образом, массивы обеспечивают не только произвольный доступ, но и быстрый последовательный!

Сортировка вставками

Допустим, у вас на компьютере записана музыка и у каждой музыке есть свой счётчик воспроизведения. Вы хотите отсортировать её по популярности: от самой часто прослушиваемой до почти не прослушиваемой.

Одно из возможных решений - пройти по списку, найти самую популярную песню и записать её в новый массив.

А теперь попробуем оценить такой алгоритм с точки зрения "О-большого". Всего у нас n песен. Чтобы найти самую популярную песню, нужно пройтись по всему массиву, то есть воспользоваться алгоритмом $O(n)$. Такую операцию нужно будет проделать n раз. В результате, всё это потребует $O(n^2)$ времени.

Хорошие вопросы

При каждом выполнении количество элементов, которое нужно проверить сокращается, так почему же время выполнения $O(n^2)$? Это очень хороший вопрос, и ответ на него связан с ролью константы в "О-большом". Тема очень хорошо разобрана в 4 главе, но кратко объясню. Вы правы, сначала нужно будет проверить n элементов, затем $n-1, n-2, n-3, \ldots, 2, 1$. В среднем уходит $\frac{1}{2} \cdot n$. Однако роль константы в О-большом игнорируются, поэтому $O\left(n \cdot n \cdot \frac{1}{2}\right)$ заменяется на $O(n^2)$

Пример кода

Мы не будем проводить сортировку музыкального списка (но так хотелось). Вместо этого напишем нечто похожее: сортировку массива по возрастанию. Для начала нам нужна функция для нахождения наименьшего значения

PYTHON

```python
def findSmallest(arr: list) -> int:
        smallest = arr[0]
        smallest_index = 0
        for i in range(1, len(arr)):
                if smallest > arr[i]:
                        smallest = arr[i]
                        smallest_index = i
        return smallest_index
```

Теперь на основе этой функции напишем сортировку выбором:

PYTHON

```python
def sectionSort(arr: list) -> list:
        newArr = []
        copiedArr = list(arr)
        for i in range(len(copiedArr)):
                smallest = findSmallest(copiedArr)
                newArr.append(copiedArr.pop(smallest))
        return newArr
```

Упражнения

1. Допустим вы строите приложение для управления финансами. Ежедневно вы записываете все свои траты. В конце месяца вы анализируете расходы. При работе с приложением выполняется множество операций вставок и небольшое количество операций чтения. Какую структуры вы будете использовать?

2. Допустим, вы пишите приложение ждя приёма заказов от посетителей ресторана. Приложение должно хранить список заказов. Официанты добавляют заказы в список, а повора читают заказы из списка и готовят их. Заказы образуют очередь: официанты добавляют заказы в конец очереди, а повар берёт первый заказ из очереди и начинает готовить. Какую структуру данных вы бы использовали: массивы или связанные списки? Почему?
3. Проведём мысленный эксперемент. Допустим, Facebook хранит список имён пользователей. Когда кто-то пытается зайти на сайт система пытается найти имя пользователя. Если имя в списке, то пользователь может войти. Пользователи приходят на Facebook достаточно часто, поэтому поиск выполняется достаточно часто. Будем считать, что Facebook использует бинарный поиск. Как бы вы реализовали список пользователей: через список или массив? (Подсказка: вспомните, какой доступ нужен бинарному поиску)
4. Пользователи также довольно часто создают новые учётные записи на Facebook. Предположим, вы решили использовать массив для хранения списка пользователей. Какими недостатками обладает массив для выполнения вставки? Если вы используете бинарный поиск, то, что произойдёт при добавлении новых пользователей в массив?